Quantstamp Security Assessment Certificate

October 29th 2020 — Quantstamp Verified

SKALE Allocator

This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type	Token Vesting Contracts
Auditors	Ed Zulkoski, Senior Security Engineer Kevin Feng, Software Engineer Kacper Bąk, Senior Research Engineer
Timeline	2020-08-05 through 2020-08-12
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	SKALE SAFT Core
Documentation Quality	High
Test Quality	Undetermined

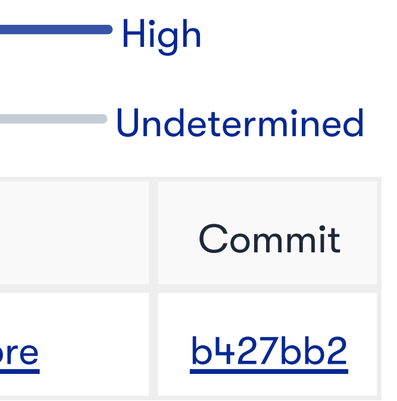
Source Code

Repository	Commit
skale-saft-core	b427bb2

Goals

- Do the smart contracts implement the provided specification?
- Can funds be locked or stolen?
- Do vesting schedules unlock funds at the appropriate time?

Total Issues	**9** (7 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**2** (2 Resolved)
Low Risk Issues	**3** (3 Resolved)
Informational Risk Issues	**3** (1 Resolved)
Undetermined Risk Issues	**1** (1 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

○ Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
○ Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Quantstamp has reviewed the SAFT, Core, and associated smart contracts. The code is generally well-documented and specified. During the audit, several issues were found, including several functions that were incomplete (containing TODO statements), some functions that may not behave correctly for corner-cases, and issues related to function parameter checks. We recommend addressing all issues before deployment. Further, although test coverage was quite high, we recommend additional testing to ensure edge-cases are handled properly, particularly for complicated functions such as `getTimeOfNextUnlock()`.
Update: All fixes from the Skale team have been reviewed and marked resolved based on commit 6c64026.

ID	Description	Severity	Status
QSP-1	Unresolved TODOs	^ Medium	Fixed
QSP-2	Incorrect Unlock Times in `getTimeOfNextUnlock()`	^ Medium	Fixed
QSP-3	Stub `getAndUpdateForbiddenForDelegationAmount()` function	ˇ Low	Fixed
QSP-4	Unclear semantics of `addSAFTRound()`	ˇ Low	Fixed
QSP-5	Missing check in `withdrawBounty()` and `requestUndelegation()`	ˇ Low	Fixed
QSP-6	Misusing `require()`/`assert()`/`revert()`	○ Informational	Acknowledged
QSP-7	Privileged Roles and Ownership	○ Informational	Acknowledged
QSP-8	Unchecked address parameters	○ Informational	Fixed
QSP-9	Unclear `delegate()` functionality	? Undetermined	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.6.12
- Mythril v0.22.8

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`s
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth -x path/to/contract`

Findings

QSP-1 Unresolved TODOs

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Core.sol`, `CoreEscrow.sol`, `SAFT.sol`

Description: There are several TODO statements that still exist in the code:

1. In SAFT.sol#199 - `// TOOD: Fix index error`
2. In Core.sol#192 - `// TODO add deactivate logic`. Related, the function does not change the `HolderStatus` to `TERMINATED`.
3. In Core.sol#219 - `// TODO: Remove to allow both past and future vesting start date`
4. In Core.sol#372 - `TODO: remove, controlled by Core Escrow`
5. In CoreEscrow.sol#240 - `TODO: missing moving Core holder to deactivated state?`
6. In several locations - `replace with ProxyFactory when @openzeppelin/upgrades will be compatible with solidity 0.6`

Recommendation: Ensure that the intended functionality is added for each TODO.

Update: this has been resolved with the exception of bullet 6, which is pending updates to OpenZeppelin.

QSP-2 Incorrect Unlock Times in `getTimeOfNextUnlock()`

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `Core.sol`, `SAFT.sol`

Description: Consider a SAFT that lasts 2 years, vests every 1 year, and has no lockup period. Consider a `SAFTHolder` with a `startVestingTime = "Dec. 30, year 0"`, and this function is invoked on "Jan 1, year 1".
The local variables in `getTimeOfNextUnlock()` will be assigned as follows:

- `dateTime = year 1`
- `lockupTime = year 0`
- `finishTime = year 2`
- `numberOfDonePayments = 1`
- `numberOfAllPayments = 2`

Since `numberOfAllPayments <= numberOfDonePayments + 1`, the check on L325 will succeed, and the function will incorrectly report Dec. 30, year 2 as the next unlock time, instead of Dec 30, year 1.
As an alternative scenario, if instead of a 2 year SAFT, it is 3 years, then L325 will fail, and we correctly have the following:

- `nextPayment = 1`
- returns "Dec. 30, year 1"

Note that the same functionality exists in `Core._getNumberOfCompletedUnlocks()`.

Recommendation: Revise the functionality of these two functions. Add new test cases corresponding to edge-cases, such as dates near the end or start of a year.

QSP-3 Stub `getAndUpdateForbiddenForDelegationAmount()` function

Severity: *Low Risk*

Status: Fixed

File(s) affected: `SAFT.sol`

Description: The function `getAndUpdateForbiddenForDelegationAmount()` is a stub. It also contains the comment `network_launch_timestamp` which is unclear.

Recommendation: Implement the function if necessary, and clarify the meaning of the comment.

QSP-4 Unclear semantics of `addSAFTRound()`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `Core.sol`, `SAFT.sol`

Description: It is not clear why `vestingPeriod` is not considered when sanitizing the function arguments, since the first two parameters are in months, but the `vestingTimes` parameter could be days, months, or years. For example, if `vestingPeriod == 3` which corresponds to years, then an input of `lockupPeriod == 1 (month)`, `fullPeriod == 2 (month)`, `vestingTimes == 1 (year)` would pass the checks.
The same issue exists in `Core.addCore()`.

Recommendation: Revise the require-statements ensuring the `vestingPeriod` is taken into consideration.

QSP-5 Missing check in `withdrawBounty()` and `requestUndelegation()`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `CoreEscrow.sol`

Description: In the comment block, it states "Only Owner can withdraw bounty to Core contract after Core holder is deactivated.". However, there is no check that the holder is deactivated if the caller is the contract owner.
A similar issue exists for `requestUndelegation()`, which states "Only Owner can request undelegation after Core holder is deactivated (upon holder termination)."

Recommendation: Add a require-statement ensuring that the account is deactivated if the owner is the caller in each function.

QSP-6 Misusing `require()`/`assert()`/`revert()`

Severity: *Informational*

Status: Acknowledged

File(s) affected: `BokkyPooBahsDateTimeLibrary.sol`

Description: `require()`, `revert()`, and `assert()` all have their own specific uses and should not be switched around.

- `require()` checks that certain preconditions are true before a function is run.
- `revert()`, when hit, will undo all computation within the function.
- `assert()` is meant for checking that certain invariants are true. An `assert()` failure implies something that should never happen, such as integer overflow, has occurred.

Recommendation: In `BokkyPooBahsDateTimeLibrary.sol`, use `assert` instead of `require` on lines `217`, `232`, `236`, `240`, `244`, `248`, `262`, `277`, `281`, `285`, `289`, and `293`

QSP-7 Privileged Roles and Ownership

Severity: *Informational*

Status: Acknowledged

File(s) affected: `Core.sol`, `SAFT.sol`

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract.
In the Core and SAFT contracts, the owner is required to start all vesting plans, and may terminate a vesting holder at any point.

Recommendation: Although this centralization seems natural for this type of vesting contract, users (holders) should be made aware of the roles of the owner in each contract through documentation.

QSP-8 Unchecked address parameters

Severity: *Informational*

Status: Fixed

File(s) affected: `CoreEscrow.sol`

Description: In `initialize()`, the address parameters are not checked to be non-zero. This may lead to incorrect initialization if the default values are unintentionally passed during deployment.

Recommendation: Add require-statements ensuring that each address argument is non-zero in `initialize()`.

QSP-9 Unclear `delegate()` functionality

Severity: *Undetermined*

Status: Fixed

File(s) affected: `CoreEscrow.sol`

Description: In `delegate()`, if the holder has already delegated their full vested amount to a validator, but then invokes the function again with a different `validatorId`, what is the result? Will `delegationController.delegate(validatorId, amount, delegationPeriod, info);` fail?

Recommendation: Clarify this functionality with added documentation.
Update from Skale team: delegate() logic is handled and checked in `skale-manager` smart contracts (`delegationController.sol`), `ESCROW.sol` call is a proxy layer for beneficiaries, though not covered with error handling additionally.

Automated Analyses

Slither

Slither reported no issues.

Mythril

In `SAFT.getTimeOfNextUnlock()`, Mythril warns that an assertion (i.e., array bounds check) may fail on the following line: `SAFTRound memory saftParams = _saftRounds[saftHolder.saftRoundId - 1];`. Ideally, the function should first check that the `saftHolder` exists before this array access, however since this is an external view function, there is no gas-cost implications to this assert.

A similar issue was found in `Core.getVestingCliffInMonth()` and `Core.calculateVestedAmount()`.

Update: resolved. From the Skale team: the indexes in the functions above are taken from the contract's memory (not externally) thereby are correct and not checked additionally.

Adherence to Specification

The code is well specified. Some discrepancies between the specification and the code were already noted in the findings above.

Code Documentation

The code is generally well documented. Some minor issues:

1. "An core" should be changed to "a core" throughout.
2. In the function `CoreEscrow.retrieveAfterTermination()` the comment block has the requirement "Core must be active". The core should be *not* active. Further, the corresponding require-statement in the function should state ""Core holder is still active".

Update: all issues have been resolved.

Adherence to Best Practices

1. Favor using `uint256` instead of `uint` in order to make the size of integer variables explicit.
2. There is commented out code that should be removed:
    1. SAFT.sol#79 -- `// SAFTRound[] private _otherPlans;`
    2. SAFT.sol#84-85 -- related to `_holderToEscrow`
    3. SAFT.sol#213-217 in `connectHolderToSAFT()`
    4. SAFT.sol#364 in `initialize()`
    5. SAFT.sol#404-406 in `_getNumberOfCompletedUnlocks()`
    6. CoreEscrow.sol#108 in `retrieve()`
    7. Core.sol#85 -- `// mapping (address => uint) private _vestedAmount;`
    8. Core.sol#218-219 in `connectHolderToPlan()`
    9. Core.sol#383-388 the function `getLockedAmountForDelegation()`
3. Some functions such as `_addMonthsAndTimePoint()` , `_getTimePointInCorrectPeriod()`, and `_getPartPayment()` are cloned across multiple contracts, and could be abstracted into a library.
4. In `SAFT.addSAFTRound`, `vestingPeriod` could be given an enum type. Similarly for `Core.addCore()`.
5. The function `SAFT.getAndUpdateLockedAmount()` does not actually update anything. Consider renaming.
6. The function `SAFT.getAndUpdateForbiddenForDelegationAmount()` does not actually update anything. Consider renaming.
7. In `SAFT.getTimeOfNextUnlock()`, the variable `lockupDate` is declared twice.
8. In `SAFT.getTimeOfNextUnlock()`, the expression `timeHelpers.addMonths(saftHolder.startVestingTime, saftParams.lockupPeriod)` is computed twice (L310, L316).
9. In `Core.approveHolder()` on L118-119: the check in L118 is unnecessary since `UNKNOWN != CONFIRMATION_PENDING`.

Update: all issues have been resolved.

Test Results

Test Suite Results

```
  Contract: Allocator
     ✓ should register beneficiary (786ms)
     ✓ should get beneficiary data (978ms)
     ✓ should not start vesting without registering beneficiary (188ms)
     ✓ should start vesting with registered & approved beneficiary (940ms)
     ✓ should stop cancelable vesting after start (2903ms)
     ✓ should not stop uncancelable vesting after start (2645ms)
     ✓ should not register Plan if sender is not a vesting manager (121ms)
     ✓ should not connect beneficiary to Plan  if sender is not a vesting manager (210ms)
     ✓ should not register already registered beneficiary (930ms)
     ✓ should not register Plan if cliff is too big (115ms)
```

```
    ✓ should not register Plan if vesting interval is incorrect (116ms)
    ✓ should not connect beneficiary to Plan if amounts incorrect (186ms)
    ✓ should be possible to delegate tokens in escrow if allowed (1556ms)
    ✓ should allow to retrieve all tokens if beneficiary is registered along time ago
(1430ms)
    ✓ should operate with fractional payments (1753ms)
    ✓ should correctly operate Plan 4: one time payment (1939ms)
    ✓ should correctly operate Plan 5: each month payment (4177ms)
    ✓ should correctly operate Plan 5: each 1 day payment (6449ms)
    ✓ should correctly operate Plan 5: each 1 year payment (3876ms)
    ✓ should correctly operate Plan 6: each day payment for 3 month (45261ms)
    ✓ should correctly operate Plan 7: twice payment (1625ms)
    ✓ should not add plan with zero vesting duration (180ms)
  when beneficiary delegate escrow tokens
     ✓ should be able to cancel pending delegation request (349ms)
     ✓ should be able to undelegate escrow tokens (382ms)
     ✓ should allow to withdraw bounties (960ms)
  when Plans are registered at the past
     ✓ should unlock tokens after lockup (177ms)
     ✓ should be able to transfer token (1039ms)
     ✓ should not be able to transfer more than unlocked (1126ms)
     ✓ should unlock tokens first part after lockup (200ms)
  when all beneficiaries are registered
     ✓ should show balance of all escrows (212ms)
     ✓ All tokens should be locked of all beneficiaries (367ms)
     ✓ After 6 month (455ms)
     ✓ After 9 month (549ms)
     ✓ After 12 month (725ms)
     ✓ should be possible to send tokens (3943ms)
     ✓ After 15 month (545ms)
     ✓ After 16, 17, 18 month (1428ms)
     ✓ After 24, 30, 36 month (1114ms)
  should calculate next vest time correctly
     ✓ from Dec 30, year based vesting (563ms)
     ✓ from Dec 30, month based vesting (617ms)
     ✓ from Dec 30, day based vesting (717ms)


  41 passing (5m)
```

Code Coverage

The code is well-covered by the test suite.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	97.08	69.64	100	97.06	
Allocator.sol	96.58	79.17	100	96.4	371,440,461,487
Escrow.sol	97.78	53.13	100	97.92	69
Permissions.sol	100	50	100	100	
contracts/interfaces/	100	100	100	100	
IContractManager.sol	100	100	100	100	
ITimeHelpers.sol	100	100	100	100	
contracts/interfaces/delegation/	100	100	100	100	
IDelegationController.sol	100	100	100	100	
IDistributor.sol	100	100	100	100	
ITokenState.sol	100	100	100	100	
contracts/interfaces/openzeppelin/	100	100	100	100	
IProxyAdmin.sol	100	100	100	100	
IProxyFactory.sol	100	100	100	100	
All files	**97.08**	**69.64**	**100**	**97.06**	

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
4d0ae9cfe821957af39ca515b6ff4c0c3898f61370b51fb2b41500433bcc0bc7
./contracts/Allocator.sol

2fcc8e6a54365d2519dac0fd7ae984f68f75f8ff5634ebf83b647c99560efd2a
./contracts/Permissions.sol

b5332bbf2bff34f1dac359f825799dce5ce531d9417250a17c105b7862cd2192 ./contracts/Escrow.sol

03ffbd8b46d52e1df026870a9b9c9e73c7498268a509abf55f3c9a0260eb8517
./contracts/interfaces/IContractManager.sol

f9619b00864d00de49e55e790be5982fb5b8129b46b89ccba905e382fc51fe8a
./contracts/interfaces/ITimeHelpers.sol

c664c817f9821fc45f891f0af03ea082bfd6d3fa0d9d4fcf69e038aec8e85b2d
./contracts/interfaces/openzeppelin/IProxyFactory.sol

618b203ab75e363e55497680eeb4969ea8e4e9c7fa8e79b0b0b71d14007ddd49
./contracts/interfaces/openzeppelin/IProxyAdmin.sol

a3bfc5a38b3caf625e25d6654cc5dafc67830e0dd11823d001f41afc380448c0
./contracts/interfaces/delegation/ITokenState.sol

37bf9c07812b1adfba8c4811ef6b1613a3b4c94b3ea53787ad51c9273a2785dd
./contracts/interfaces/delegation/IDistributor.sol

0652c9818cdb040026bad6011b586d9780b4f3675d84c54c2933d5c793f56182
./contracts/interfaces/delegation/IDelegationController.sol

b5faaaac59256e0facd0ded27cbe1920d91e6ca552bf8a4f5eb07179f07d407f
./contracts/test/LockerMock.sol

b051726dfdf768c0f6f298126ecedebaa50bdfd5daac4e5859f27604797c13b2
./contracts/test/ContractManager.sol

28197c20c7e62aadab363afa20999e46aec4500031e2f3e123efbe1e5b3e436a
./contracts/test/ConstantsHolderMock.sol

d5821e0e489b04360e1230e7d8319d5cea01fa71025528eb1a2bc2bf5bb1f466
./contracts/test/ProxyFactoryMock.sol

a8f56126ad4ad8c67e063a641b5717a451e63d5990cf4ecb88d85b1e0f7d8ac2
./contracts/test/TokenStateTester.sol

9d75e4fd92df7af8f922848cc513ac2483ad06e5e0c70c42c276e0d4b10270e0
./contracts/test/SkaleTokenTester.sol

c1d797b8788804140f58b57c73676fabf466657ae5f64fe69bfa2889ab0f4917
./contracts/test/DistributorMock.sol

b7b68bc65367ce19b9da7a6212e92d9d9deaf29449863c4a89db6613955ba493
./contracts/test/TimeHelpersTester.sol

11b0131aadafc32afc4c113652525e8f128527c1bbc73fbd25b96cd27fd2cd3d
./contracts/test/TokenLaunchManagerTester.sol

a2106e94e189039dcf341e9bb98c69e77ae8b61f073ad214401592e0b87d7c08
./contracts/test/DelegationControllerTester.sol

1bfe6a9fa226689d24996f522dfffb250539a78d188de278f85d4ac1e063e4c7
./contracts/test/interfaces/ILocker.sol

51b24273656b8053643df601c6d1c4dd1ecbe0f5a2235550f390838904b4b682
./contracts/test/thirdparty/Migrations.sol

251cf29182e47b7fdc593e6b0a2923587f921d319d859e3fb1eb91159b7efa2f
./contracts/test/thirdparty/BokkyPooBahsDateTimeLibrary.sol
```

13a0f044718497ef82e2339f3e739dffb50e858c9974fb4326bc23ae17505a48
./contracts/test/utils/StringUtils.sol

Tests

6a8e4d898d2ef3f0f879788f7f2fc369ca76ae0b08764815d14b9a893f04f242 ./test/Allocator.ts

6e7867b7198b66578a6dc32c8fe21e02bc069150ef272182ba6d23c618f7a280 ./test/tools/types.ts

e58228b8f7b3eca303823d7c2d33156b3f12a413676022f66c1c3474e799c828
./test/tools/vestingCalculation.ts

90c2bdf776dc095f4dd0c1f33f746bea37c81b3ad81d51571d8dc84d416ecd13 ./test/tools/elliptic-types.ts

361f1be518e213e9106378a90463ad8bb673f81ed1d2505602d12a3485a9ba52
./test/tools/command_line.ts

dbe983ad378eb15da090040af997b3ebb5cf2a1bd01b819a900f33886eb739b3 ./test/tools/time.ts

990ba91e6946ce16a6e087707238a0b50a6fbe9596e9f786ad2ee14cc3ef4519
./test/tools/deploy/allocator.ts

035dc4d0efdde1948276f69ac00960dc4b3e2f1f32fef7792e11d797646b5e62
./test/tools/deploy/escrow.ts

703d2e51da3fe0426847cc1d8ced2c707b50c5d6e9b67e0d43b8ef102458012e
./test/tools/deploy/contractManager.ts

de7143ec7676eb978f26e9c8bb55ac861a43d4b4089ab7701d71e42335943881
./test/tools/deploy/factory.ts

3d92b6c89781dfd476d1971283951bdb5f6941aa5148865ad87e535e4c26da8d
./test/tools/deploy/test/tokenStateTester.ts

ef46653cbd3ea549621e67528508f49bae491c42f31dcef7b52e7f4b1ba3a829
./test/tools/deploy/test/tokenLaunchManagerTester.ts

269bc5b375f09b181e5a3b4c7ef8359c397146381bc230af1e8d97f9225197f3
./test/tools/deploy/test/proxyFactoryMock.ts

e2f571b76e72739b2052eccdff5705f8a06d213f78318e055c028c8f42753be2
./test/tools/deploy/test/skaleTokenTester.ts

9d227e2ce4b26348f4ec02b265d283102fe0a13ad75f5746b0accd8d115a531d
./test/tools/deploy/test/timeHelpersTester.ts

32bb05d05ff5171a33bba4827171d22c2cf70fa2daa65e66d7e0869b6806dbf4
./test/tools/deploy/test/delegationControllerTester.ts

492f16224e404a5bfc2610b34041449b257e06f9f6f9680de5acc5a67351f14d
./test/tools/deploy/test/constantsHolderMock.ts

Changelog

- 2020-08-12 - Initial report
- 2020-09-10 - Revised report based on commit 6c64026

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or

ct or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.